Fiche TD12 : Concurrence

Exercice 1 (*) *Concurrence bancaire*

Deux agences d'une banque, l'une à Nancy, l'autre à Tours veulent mettre à jour le même compte bancaire. Pour ce faire, l'agence de Nancy effectue les opérations suivantes :

1. courant $\leftarrow$ récupérer le montant sur le compte $18A3$
2. nouveau $\leftarrow$ courant $+1000$
3. montant sur le compte $18A3 \leftarrow$ nouveau

L'agence de Tours quant à elle effectue les opérations suivantes :

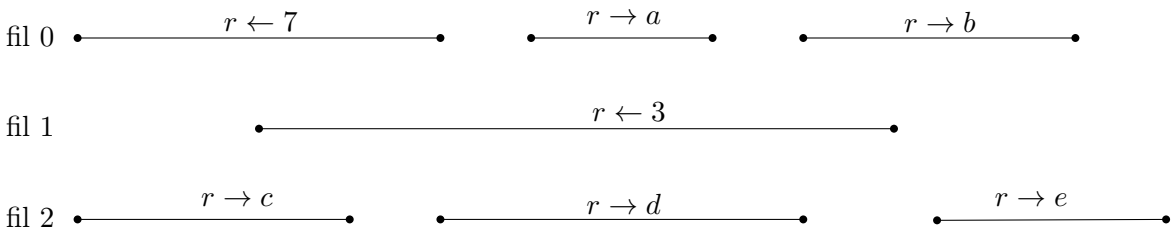
A. courant $\leftarrow$ récupérer le montant sur le compte $18A3$
B. nouveau $\leftarrow$ courant -1000
C. montant sur le compte $18A3 \leftarrow$ nouveau

1. Montrer qu'il est possible que le montant sur le compte $18A3$ ait augmenté ou diminué de 1000.
2. Si on suppose que l'agence de Nancy commence en premier à exécuter son code, quel est le montant sur le compte $18A3$ à la fin des deux transactions ?
3. Comment faire pour garantir qu'à la suite de ces deux transactions le montant sur le compte n'ai pas changé ?

Exercice 2 (**) *Entrelacement*

On considère trois fils partageant une variable entière r initialement nulle. Ces fils écrivent et lisent la variable r : on note $r \leftarrow x$ si le fil écrit x dans r et $r \rightarrow x$ si le fil lit la variable r pour la stocker dans x.

On suppose que chaque lecture ou écriture est atomique et prend place durant un certain intervalle de temps ; ces derniers étant symbolisés sur le diagramme suivant :



1. Que peut valoir c ?
2. Que peut valoir a ?
3. Que peut valoir e ?
4. Si c vaut 7, que peut valoir d ?
5. Si d vaut 0, que peut valoir c ?
6. Si a vaut 3, que peut valoir e ?

Exercice 3 (***) *A en perdre le compte*

On considère dans un premier temps le pseudo code suivant :

```
compteur ← 0

incremente ( ) =
    i ← 0
    Tant que i < 10
        i ← i + 1
        compteur ← compteur +1
```

On suppose que deux fils, notés A et B exécutent la fonction incremente en parallèle (la variable compteur est donc une variable globale partagée par les deux fils).

1. Montrer qu'en fin d'exécution le compteur peut valoir 20.
2. Montrer qu'il peut valoir 10.
3. Montrer qu'il peut valoir 2.

On modifie le code précédent et on fait exécuter cette nouvelle version en parallèle par nos deux fils :

```
compteur ← 0
i ← 0

incremente ( ) =
    Tant que i < 10
        i ← i + 1
        compteur ← compteur +1
```

4. Quelle est la plus petite valeur possible que peut contenir compteur en fin d'exécution ? Et la plus grande ? *Remarque : Venez me proposer vos réponses pour vérification, cette question est fourbe.*

Exercice 4 (*) *Autour de l'algorithme de Petersen*

On considère les algorithmes suivants destinés à implémenter un mutex dans le cas où on a deux fils. On utilise pour ce faire un tableau `want` de booléens indiquant en case i si le fil i souhaite acquérir le verrou et d'une variable `turn` indiquant quel fil a la priorité.

```
create_lock ( ) =
    Renvoyer {turn = 0, want =[false, false]}

lock(mutex, thread) =
    mutex.want[thread] ← true
    mutex.turn ← thread
    other ← 1-thread
    Tant que mutex.turn = other et mutex.want[other], rien

unlock(mutex, thread) =
    mutex.turn ← 1-thread
    mutex.want[thread] ← false
```

1. Montrer qu'un verrou ainsi implémenté ne vérifie pas l'exclusion mutuelle.
2. Cet algorithme est très similaire à l'algorithme de Petersen qui lui garantit l'exclusion mutuelle. Où se trouve la différence ?

On considère à présent que les fonctions create_lock, lock et unlock sont celles de l'algorithme de Petersen.

3. Dans chacun des cas suivants, justifier que l'affirmation est vraie ou fournir une trace d'exécution qui montre qu'elle ne l'est pas :

   (A) Si le fil T_0 rentre dans lock avant le fil T_1 (c'est-à-dire exécute la première instruction de lock avant que T_1 ne le fasse), alors T_0 obtiendra le verrou avant T_1.

   (B) Si T_0 exécute toutes les instructions qui précèdent la boucle tant que de lock avant que T_1 n'exécute la première instrction de lock alors T_0 obtiendra le verrou avant T_1.

   (C) Si T_0 et T_1 cherchent tous les deux à acquérir le verrou, alors aucun des deux fils ne peut l'obtenir deux fois de suite.

4. Que vous inspirent les réponses à la question précédente ?

Exercice 5 (**) *Interblocage et sémaphores*

On note P (= acquire = wait) la demande sur un sémaphore et V (= release = post) sa libération. Dans chacune des situations suivantes, dire si il est possible d'obtenir un (inter)-blocage :

1. a et b sont deux sémaphores dont les compteurs valent respectivement 1 et 0 au début. Deux fils exécutent de manière parallèle les instructions suivantes :

   - Fil 1 : P(a), V(a), P(b), V(b).
   - Fil 2 : P(a), V(a), P(b), V(b).

2. a,b et c sont trois sémaphores dont les compteurs valent initialement tous 1. Trois fils exécutent de manière parallèle les instructions suivantes :

   - Fil 1 : P(a), P(b), V(b), P(c), V(c), V(a).
   - Fil 2 : P(c), P(b), V(b), V(c), P(a), V(a).
   - Fil 3 : P(c), V(c), P(b), P(a), V(a), V(b).

3. a,b et c sont trois sémaphores dont les compteurs valent initialement tous 1. Deux fils exécutent de manière parallèle les instructions suivantes :

   - Fil 1 : P(a), P(b), V(b), P(c), V(c), V(a).
   - Fil 2 : P(c), P(b), V(b), V(c).

Exercice 6 (**) *Lecteurs-rédacteurs*

Le problème des lecteurs-rédacteurs est le suivant. On dispose d'une base de données à laquelle deux types d'utilisateurs ont accès. Les lecteurs ne font que lire des informations dans la base. Les rédacteurs peuvent lire ou écrire dans la base. On impose les contraintes suivantes :

- Plusieurs lecteurs peuvent consulter la base en même temps.
- Si un rédacteur est en train d'utiliser la base, aucun autre utilisateur (ni lecteur, ni un autre rédacteur) ne peut l'utiliser en même temps.

On propose d'utiliser le code suivant pour régir le comportement des lecteurs et des rédacteurs : tous partagent deux sémaphores lecteur et redacteur initialisés à 1 et une variable nb_lecteurs valant initialement 0. Puis :

Code pour chacun des lecteurs :

```
P(lecteur)
nb_lecteurs ← nb_lecteurs +1
Si nb_lecteurs = 1
    P(redacteur)
V(lecteur)
Faire des lectures
P(lecteur)
nb_lecteurs ← nb_lecteurs −1
Si nb_lecteurs = 0
    V(redacteur)
V(lecteur)
```

Code pour chacun des rédacteurs :

```
P(redacteur)
Faire des lectures et écritures
V(redacteur)
```

1. Justifier que lorsqu'un rédacteur accède à la base il en a l'accès exclusif.
2. Justifier que plusieurs lecteurs peuvent accéder à la base en même temps.
3. Le comportement des fils serait-il impacté si on utilisait un mutex plutôt qu'un sémaphore binaire pour lecteur ? Même question pour redacteur.
4. Montrer qu'avec ce code il peut y avoir famine pour les rédacteurs.
5. Ecrire un pseudo-code corrigeant ce problème de famine en utilisant un sémaphore supplémentaire.

Exercice 7 (**) *Organisation de tâches à l'aide de sémaphores*

On considère l'exécution parallèle de trois processus P1, P2, P3 ayant les caractéristiques suivantes :

- P1 exécute en boucle les tâches A puis B.
- P2 exécute en boucle les tâches U puis V.
- P3 exécute en boucle la tâche X.

De plus, les contraintes suivantes doivent être respectées :

(1) Les tâches B et U sont en exclusion mutuelle.

(2) La tâche A produit un élément (qui sera consommé) nécessaire à la complétion de la tâche X. Autrement dit, l'occurrence i de X ne peut pas commencer avant la fin de l'occurrence i de A.

1. On note dA_i le début de la i-ème occurrence de la tâche A et fA_i sa fin (similairement pour les autres tâches). Les traces d'exécution suivantes sont elles possibles ? Si non, indiquer tous les endroits qui posent problème.
   a) $dA_1, fA_1, dX_1, dB_1, dU_1, fX_1, fU_1, fB_1, dA_2, dX_2, dV_1, fV_1, fX_2, fA_2, dU_2, dB_2, fU_2, fB_2$.
   b) $dA_1, fA_1, dX_1, dB_1, fB_1, dA_2, dU_1, fA_2, fX_1, fU_1, dX_2, dV_1, fV_1, fX_2, dU_2, fU_2, dB_2, fB_2$.
2. Expliquer comment utiliser un sémaphore pour garantir la contrainte (1).
3. Expliquer comment utiliser un sémaphore pour garantir la contrainte (2).
4. Peut-on utiliser le même sémaphore pour garantir en même temps les conditions (1) et (2) ? Justifier.